Control d'accés al *Routing Information Tree* en un esquema de xarxes DTN actives

Ramsés Pascual Clemente

Resum— En aquest treball es modifica una variant del ja conegut sistema de control d'accés anomenat *ABAC*. Aquesta variant s'implementa dins d'una *DTN* activa, permetent així afegir un component de seguretat dins la xarxa. Al ser un tipus de xarxa que permet endarreriments de la transmissió, *ABAC* ha de ser adaptat per aquest cas, variant els mòduls interns del que es composa, evitant d'aquesta manera que hi hagi pèrdua de dades per falta de connectivitat. Al llarg de l'article es pot observar: l'entorn en que s'aplicarà aquest TFG; els petits canvis que es realitzaran, tant d'*ABAC*, que per aquest projecte els seus mòduls no estaran directament connectats, com dels components de la xarxa; petites variacions en el codi i l'agregació d'algunes comparacions per fer servir el filtre proposat; i finalment, la metodologia utilitzada per arribar a la solució proposada, juntament amb la conclusió extreta després de trobar una de les possibles respostes al problema.

Paraules clau—Attribute Based Access Control (ABAC), Acces Control List (ACL), Sistema de Control d'Accés (SCA), Policy Enforcement Point (PEP), Policy Decision Point (PDP), Delay Tolerant Network (DTN)

Abstract—In this work, a modification of the already well known access control system ABAC is presented. Implementing this variant inside an active DTN, allowing to add a security component into the network. As DTN allows transmission delays, ABAC had to be adapted in order to work correctly with this case. Changes its internal modules have been preventing leaks in case of connectivity losses. Along the present paper, it may be observed: the environment where the TFG will be applied; the changes implemented, both in ABAC, which modules will not be connected directly, as well as in the network components, some code variations and the integration of some comparisons in order to use the ABAC's filter; and finally, the methodology used to find the proposed solution, together with the conclusion after reaching a possible answer to the problem.

Index Terms—Attribute Based Access Control (ABAC), Acces Control List (ACL), Access Control System (ACS), Policy Enforcement Point (PEP), Policy Decision Point (PDP), Delay Tolerant Network (DTN)

- *E-mail de contacte: ramsespascual@gmail.com*
- *Menció realitzada: Tecnologies de la Informació.*
- *Treball tutoritzat per: Sergi Robles (DEIC)*
- *Curs 2013-14*

1 Introducció

Un dels interessos de recerca del grup SeNDA (Security of Network and Distributed Aplications)[7], és la modificació del concepte de *DTN (Delay Tolerant Network)* [9] per crear *aDTN* (*active DTN*).

DTN és un protocol de tipus de xarxa que tolera les pèrdues o retrassos en la connexió amb la resta de nodes. Al permetre aquests inconvenients quan es tracta de nodes en moviment, és una bona alternativa a les connexions tals com el *3G* o *WiMAX*, els quals són més costosos o necessiten components més sofisticats als disponibles.

La variant del protocol DTN, anomenada *aDTN*, utilitza biblioteques creades explícitament per la comesa i les necessitats existents del projecte en el que treballen.

aDTN utilitza un *RIT (Route Information Tree)* per a conèixer les posicions dels altres components de la xarxa, i transmetre informació entre els seus components. *aDTN* permet que qualsevol usuari es pugui connectar i participar.

Per aconseguir accés al *RIT* es fa servir un codi específic dins d'una *API* creada pel mateix grup. L'*API* (*Application Programming Interface*) serveix per facilitar als programadors el procés que han de seguir per aconseguir el propòsit del codi. Conté les biblioteques que especifiquen les rutines, estructures de dades, les classes i les variables.

L'accés al *RIT* pot ser realitzat per qualsevol component de la xarxa, i no té cap tipus de seguretat, és a dir, que no hi ha control a l'hora de realitzar les accions contra els recursos. Per tant, un intrús pot entrar i modificar, adquirir, inserir o eliminar dades. Aquest TFG es proposa trobar una de les possibles solucions a aquest problema, que utilitza un Sistema de Control d'Accés (SCA) per posar-hi remei.

El propòsit d'un SCA situat dins d'una xarxa és protegir objectes, dades, accés a recursos o qualsevol altre tipus de tecnologia de la informació d'operacions no autoritzades. Es podria veure que és com un porter, que segons unes característiques de l'usuari, realitza per l'usuari les tasques, i aquest retorna la informació. El porter fa les accions seguint una llista de regles les quals consulta abans de fer res.

L'objectiu principal d'aquest TFG és limitar l'accés del codi contingut als *bundles*, cap al RIT. Els *bundles* contenen el codi per realitzar les accions que l'usuari vol fer. Per a aconseguir-ho, l'opció escollida va ser incorporar un SCA a *aDTN*, per incrementar la seguretat en el sistema.

Aquest objectiu es va dividir en 4 subobjectius. Aquests estan resolts en les tres següents seccions. Estan ordenats cronològicament en el temps, tal qual es van realitzar durant el desenvolupament del TFG.

El primer dels objectius a resoldre va ser l'estudi de mecanismes existents a la literatura dels SCA. Aquests van acompanyats de terminologies que ajuden a comprendre els SCA. Existeixen molts SCA ja definits, i pel treball només se'n va fer servir un. Això implica l'elecció d'un d'ells. En aquest punt és on comença el segon objectiu.

El segon dels objectius, un cop complert el primer, va ser escollir, d'entre els diferents SCA existents, el que es va creure que es podria adaptar millor. Per adaptar millor es va entendre que resolia les necessitats i requisits de manera eficient. Per a fer això alguns dels SCA que es van trobar es van descartar directament, i altres es van mirar més detingudament, fins l'elecció d'un d'ells.

El tercer, un cop ja escollit el SCA a modificar per adaptar a l'*aDTN*, va ser dissenyar el mecanisme per a que els *bundles* poguessin accedir al *RIT*, sempre complint amb els requisits i les necessitats estipulats.

L'últim dels objectius va ser implementar, provar i integrar el disseny del tercer objectiu. Pel primer dels passos, es va tenir que desenvolupar l'aplicació en llenguatge C. Fins aquí es troba la informació dins de l'apartat de RITAC (*RIT Access Control*).

Dins d'aquest últim objectiu, les proves i els resultats van quedar registrats, mostrats en una secció de l'article.

Per acabar, es va buscar una manera de que la integració del TFG no suposés un gran canvi als mòduls ja existents dins l'*aDTN*. És a dir, la seva incorporació dins de la xarxa no comportés grans modificacions al disseny, a la programació i a la integració.

D'aquesta manera va ser com va néixer RITAC, fent possible tots els objectius comentats anteriorment.

La resta de l'article a partir d'aquest punt està dividida en diferentes seccions.

A la primera secció a partir d'aquí, la segona de l'article, es defineix l'Estat de l'Art, on es tractaran conceptes per posar en context al lector, tals com la definició d'una *aDTN*, diferents tipus de sistemes de control d'accés i *ABAC*.

La tercera secció de l'article és l'explicació del TFG, fent referència a tots els punts pels que ha passat l'estat d'aquests. Des de la definició de la metodologia i la planificació, passant per l'anàlisi i el disseny, la implementació i la integració dins del projecte del grup SeNDA.

La següent secció descriu els resultats i les proves realitzades durant el transcurs de la seva execució.

Per últim, hi ha les conclusions del projecte i la bibliografia utilitzada per prendre decisions.

2 Estat de l'Art

Aquest apartat està destinat a una introducció als diferents conceptes que es veuen al llarg de l'article.

No només s'explica com és i com funciona cadascun, sinó que també, resumidament, s'explica alguns dels motius de per que o no s'ha escollit aquest pel TFG.

Per començar, s'exposa i es defineix que és *aDTN*, entrant també en les *DTNs*. Cadascun dels mòduls té una petita explicació de la seva funció dins del sistema.

La segona part de l'Estat de l'Art la composa el Control d'Accés. Dins del subapartat es mostren els tipus que es van trobar que podrien haver format part del TFG en cas de complir amb totes les condicions. Estan acompanyats de l'explicació de per que si o no es va escollir.

Per últim, hi ha una breu explicació de l'estructura i el funcionament d'*ABAC*. Això inclou els mòduls i elements dels que està composat i la comunicació que farà servir.

2.1 Active DTN

Les *DTN (Delay Tolerant Network),* també anomenades *Delay and disruption-Tolerant Networks* són, tal com indica el seu nom, tolerants als retards i pèrdues de connexió.

El funcionament d'aquest protocol especialitzat en xarxes és simple. Primer de tot, tots els components de la xarxa actuen com a nodes i com a *routers* o encaminadors.

Per fer la funció de *routers*, s'utilitza una tècnica anomenada *store and forward,* així doncs, el funcionament d'aquesta es basa en la transmissió de les dades d'un node a un altre. Al no tenir una connectivitat fixa, s'han de desar fins que es puguin transmetre a un altre node.

Principalment es fan servir dos tipus de protocols per aconseguir que les dades arribin al seu destinatari. El primer es basa en enviar el missatge i esperar que aquest arribi, i el segon es basa en que la font del missatge fa vàries còpies, i les envia a diferents nodes que es trobi. La única diferència entre els dos és que el primer envia una sola còpia, mentre que el segon n'envia vàries.

Un dels possibles usos d'aquest tipus de xarxes podria ser, per exemple, una xarxa amb soroll, una xarxa que pateix un atac, o un conjunt de nodes que estan situats en un radi que no pot abastir una xarxa sense fils, i es poden moure. Aquests podrien ser alguns dels problemes per la transferència de dades en una xarxa que no tingui en constància aquests inconvenients.

DTN està sent objecte d'estudis de la *ISS* (*International Space Station*), ja que la pròpia *NASA* afirma que aquesta tecnologia encara no està realment explotada [10].

A partir d'aquest punt, és on comença *aDTN*. Aquest model fa servir les bases de *DTN*, afegint una biblioteca de funcions per poder facilitar la transferència de dades específiques entre nodes.

aDTN [1] és una variant de *DTN*. Es composa de varis mòduls:

- *Neighbour Discoverer e*stà pendent de si hi ha veïns a qui poder passar informació. Per detectar la presència de nodes veïns, cada dos segons s'envia un missatge anomenat *simple beacon message*. Un node considera que té un veí durant els quatre següents segons a rebre l'últim *beacon message*.
- L'*Executor* controla les execucions del codi.
- *Incoming Bundle Agent* el qual està al càrreg de rebre els *bundles* i possar-los a la cua.
- *Bundle Queue Manager* és el planificador de la cua dels *bundles*.
- *Outgoing Bundle Agent* s'encarrega d'enviar els *bundles*.

- *Information Tree Manager* administra l'informació guardada a l'arbre anomenat *Routing Information Tree (RIT)*. És un dels mòduls modificats en el projecte.

Aquests mòduls són els que faciliten l'ús de les propietats d'*aDTN*.

2.2 Control d'Accés

Al realitzar una connexió entre dos nodes en una xarxa o dins del propi node, els recursos dels que es disposa poden ser sensibles, és a dir, que tinguin importància i no puguin ser compartits, consultats, modificats o esborrats per qualsevol. Per això, apart dels sistemes de seguretat ja aplicats dins de la xarxa o node, es pot afegir el de control d'accés.

El control d'accés [2], com ja diu el propi nom, limita, denega o atorga accés al sol·licitant, respecte l'acció que vol realitzar sobre els recursos. Per exemple, un sol·licitant pot poder tenir permisos per obtenir dades, però no per modificar-les ni esborrar-les.

Dins d'una xarxa d'informació, és molt probable que algunes de les dades siguin públiques, i altres siguin privades. Aquest va ser el motiu pel que es dissenyen les regles que fan servir els controls d'accés. Depenent del tipus de control que es vulgui utilitzar, es fan servir diferents models de control d'accés.

Alguns dels models de control d'accés són:

- *Discretionary Access Control (DAC)* és una política determinada pel propietari de l'objecte amb el que es vol interactuar. El propietari decideix qui pot accedir a l'objecte, i amb quins permisos. Un inconvenient o avantatge de *DAC*, segons el tipus de control que es vulgui tenir, és que cadascú ha de controlar els seus propis permisos. No hi ha una unitat que ho controli tot.
- *Mandatory Access Control (MAC)* permet l'accés si i només si existeix una regla que defineixi l'accés. Es solen fer servir *sensivity labels*, cada objecte en té una. Indica el nivell de sensibilitat que té. Els nodes que tinguin accés a aquell nivell o superior podran accedir a l'objecte. *MAC* és un bon model quan s'utilitza un model en arbre per administrar l'informació.
- *Role-based Access Control (RBAC)*. La seva política és determinada pel sistema, no pel propietari. És igual que *DAC*, però sent controlat pel sistema. Un subjecte pot realitzar una operació només si ha estat assignat a aquesta o ha seleccionat el rol per a fer-la. I només podrà realitzar-la si el rol està autoritzat. Quan es sap el tipus de components del sistema, i quins permisos tindrà cadascun d'ells, és un sistema que serveix per filtrar aquesta informació. Complica la feina quan poden aparèixer diferents tipus d'usuari en qualsevol moment, ja que, o assigna permisos predeterminats i llavors els canvia si escau, o aquests no poden realitzar les tasques que pertoquen.
- *Attribute-Based Access Control (ABAC)* basa la seva política a atributs del subjecte. Defineix quines són els atributs que han de ser satisfets per a permetre l'accés. No depèn d'un permis específic, ni d'un rol. Simplement si els atributs corresponen als indicats, l'acció es podrà realitzar.

Per a aquest TFG es va escollir el sistema *ABAC*, ja que les seves facultats eren les que més s'acostaven al resultat esperat.

Per contrapartida, s'explicaran els motius pel qual no es van escollir els altres models.

DAC té les polítiques determinades pel propietari, cosa que no es vol en un sistema com el que es va implementar el treball. El propietari podria canviar els atributs necessaris estipulats a la xarxa per accedir a certes dades, per tant, els nodes que circulen, segurament, no podrien transmetre les dades.

RBAC és una possible ampliació futura del projecte cap a linies futures. Tot i que per iniciar el TFG, no és un bon model. Suposadament, qualsevol node que conegui l'API pot participar a la transferència de dades.

Per part de *MAC*, es va fer servir el seu plantejament sobre les polítiques, si no existeix la regla, no pot accedir, però dins del model *ABAC*.

ABAC va ser l'escollit ja que, dintre de les opcions que no es van descartar, era la que millor s'adapta als requisits. No és complicada d'aplicar, es pot integrar de manera que no impliqui moltes modificacions, sense necessitat de que el *kernel* o altres agents intervinguin. Manteniment i complexitat no molt elevades. Possibilitat de canvi de sistema de xifratge fàcil.

2.3 ABAC

ABAC (Attribute-Based Access Control) [3] és un sistema de control que filtra l'accés al recurs o informació basant-se en atributs que estiguin associats a un conjunt de regles o polítiques, depenent d'uns permisos prèviament establerts. El control d'accés que estableix és totalment escalable.

L'estàndart del model està composat per quatre blocs, els quals cadascun poden estar ser descrits fent servir atributs:

- Subjecte: és qui demana l'accés a la informació, creant el flux d'informació. Els atributs que descriuen el subjecte poden ser, per exemple, el rol que ocupa, la pertinença a un grup, el departament o grup al que pertany, el nivell manteniment, certificacions, competències, l'*ID* de l'usuari, etc. Aquests atributs estan inclosos dins del *token* utilitzat durant la comprovació.
- Acció: és el que el subjecte vol que es realitzi contra l'objecte, un cop autoritzada l'operació. Les més comuns solen ser llegir o escriure. En escenaris més complicats poden ser de modificació, còpia, execució, eliminació, etc.
- Recursos: identifica l'objecte implicat amb l'acció. Interpreta els atributs que identifiquen la informació. Tenen el format nom = valor.
- Context: identifica l'entorn en el que es realitza la sol·licitud. Exemples d'aquest poden ser el temps en que es realitza, la localització, el tipus de comunicació, etc. Serveix per minimitzar riscos, o crear plans de precaució.

Aquest sistema fa servir dues maneres de realitzar les consultes, *PULL* i *PUSH*.

Al model *PULL* les polítiques de control d'accés estan definides a un servidor. Les màquines que volen ser accedides per un altre agent han de fer la verificació contra el servidor, i aquest els respon. En cas afirmatiu, el procediment seguirà endavant, en cas negatiu, es denega l'accés i es talla la connexió.

PUSH és un model on les polítiques estan definides dins de les màquines que volen ser accedides. D'aquesta manera, s'estalvia la connexió al servidor. L'actualització d'aquestes polítiques és molt més complicada, ja que s'ha de fer a totes les màquines que disposin del model. Igual que en el model anterior, si la resposta és positiva, el procediment permetrà l'accés, en cas de que la resposta sigui negativa, es denega l'accés i es talla la connexió.

ABAC es composa de dos mòduls propis, anomenats *PEP* i *PDP*. Aquests dos mòduls decidir l'accés del client a les dades.

El *Policy Enforcement Point (PEP)* [4] és un component que fa que es compleixin les decisions preses pel PDP. El PEP controla l'accés a l'aplicació que conté el recurs protegit. Aquest pot estar acoblat al PDP o en una altra màquina.

Els passos que segueix durant la seva execució són:

1. El client envia una petició de recurs al *PEP*.
2. El filtre *PEP* desa la petició original del client, i crea una que anirà dirigida al *PDP*.
3. El filtre delega un missatge acord amb les polítiques de seguretat.
4. Envia la sol·licitud cap al *PDP* seguint la configuració marcades amb les polítiques.
5. El *PDP* realitza la decisió i contesta al *PEP*.
6. El *PEP* valida la resposta del *PDP*.
7. Segons la resposta del *PDP*, el filtre del *PEP* deixa passar, autoritzant la petició del client, o denega, prohibint la utilització del recurs al client.

La *Policy Decision Point (PDP)* [5] carrega les polítiques. És qui s'encarrega d'avaluar la sol·licitud que rep i respondre una de les 4 respostes possibles (*Permit, Deny, Indeterminate o Not Applicable*). Per avaluar les *queries*, consulta la informació que té sobre els nodes i els permisos i envia la resposta. En el cas del projecte actual, només es mostraran les respostes *Permit* o *Deny*.

Al mateix temps conté una extensió, anomenada *ARQ (Axiomatics Reverse Query)* que permet preguntar als nodes quins recursos pot arribar a veure.

ABAC poden funcionar sobre múltiples entorns, com *Java*, com a un servei a una aplicació d'un servidor en *J2EE*, amb *.NET* i com a aplicació *ASP* en *.NET*.

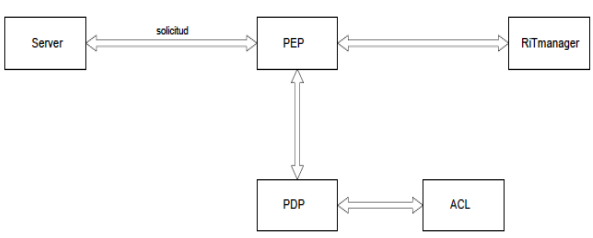

Fig1. Mostra un petit esquema del funcionament d'*ABAC*.

3 RITAC (RIT Access Control)

RITAC és la modificació realitzada al model *ABAC*, per el control d'accés al *RIT* situat dins d'una *aDTN*.

Durant les següents subseccions de l'article s'explica el transcurs del treball.

Primer de tot es presenta la metodologia i la planificació aplicada al projecte. Aquest són factors importants. Els dos influeixen a tot el projecte. El primer, donat que l'aplicació de la metodologia servirà com a guia per saber com s'ha de desenvolupar. El segon, al marcar els plaços de finalització de cada subobjectiu, i els diferents apartats dins d'aquestes.

Seguidament es mostra l'anàlisi dels SCA (Sistemes de Control d'Accés) i les condicions proposades per la seva elecció. A l'apartat corresponent també es mostra el tipus de xifratge escollit per desar les dades.

La secció dedicada al disseny explica les eleccions i les modificacions realitzades a *ABAC*. El mecanisme utilitzat per realitzar les consultes, tots els canvis fets al model original d'ABAC, els procediments que es fan durant la seva execució o els fitxers que es faran servir, incluint el nom i les dades que s'emmagatzemaran a ells, són algunes de les solucions que s'expliquen en aquest apartat.

Per acabar, es mostra com es va realitzar l'implementació i l'integració del disseny. Aquests incorporen els noms i les utilitats de cada mòdul, funció o fitxer que es fa servir.

3.1 Metodologia i planificació

La metodologia utilitzada durant el desenvolupament del projecte va ser lineal pels dos primers objectius anomenats dins la introducció, i evolutiva pels dos últims.

Els dos primers, al ser cerca d'informació i selecció d'un SCA (Sistema de Control d'Accés), es van poden fer amb un model lineal, ja que en cap cas aquests serien canviats o retocats. Com a màxim poden ser ampliats, però una bona planificació ajuda a que això no passi. Al contrari que els dos últims, que al poder-se ampliar o modificar alguna part del disseny o del codi, es va escollir un model evolutiu.

En el model evolutiu, primer de tot es comença amb disseny bàsic, en el que es realitzen les funcionalitats mínimes, desenvolupant-lo i realitzant les corresponents proves.

Un cop el funcionament del codi s'ha revisat del tot i compleix amb les expectatives, ja es pot ampliar el disseny. Tot i que en realitat es poden anar fent canvis durant el procés de proves, afegint alguna funcionalitat o ampliant l'anterior.

El model evolutiu també permet la paral·lelització d'algunes de les tasques que s'estan realitzant, i per exemple, canviar algunes parts del disseny, mentre s'estan realitzant les proves. Al mateix temps, aquest model serveix per poder mostrar en qualsevol moment, el punt en que es sitúa el projecte durant el seu desenvolupament.

La planificació, seguint el model de metodologia lineal als primers passos del projecte, i evolutiva pel

desenvolupament, permet solapaments en quan al disseny amb la programació, la programació amb les proves, i les proves amb el disseny. Tot i que aquestes sempre han de donar un petit marge de temps a l'anterior per un previ desenvolupament, ja que s'influeixen les unes a les altres.

Poder solapar tasques durant el desenvolupament afavoreix al TFG. El motiu és el fet de que si, per exemple, es descobreix algun error, possible modificació o millora que afecta al disseny durant el procés de programació o proves, aquest es pot realitzar, sense tocar cap dels altres aspectes. Tampoc és té que tornar cap a l'anterior. Si es donés l'últim cas, s'hauria de tornar un pas enrere si s'hagués utilitzat un model en cascada.

3.2 Anàlisi

La selecció d'un SCA és un procés important al afectar a la resta del treball. Si s'escull un sistema que no compleixi els requisits, el resultat d'aquest pot ser negatiu, o simplement s'hauria de refer. Per a fer-ho, donat que hi ha molts sistemes, s'han de posar certes condicions per poder descartar algunes de les opcions que es troben.

El fet d'escollir un sistema que no s'adapti com s'espera, podria afectar als plaços d'entrega. S'hauria de tornar a començar, i això endarreriria la resta del treball.

Algunes de les principals condicions són:

- Baix cost de processament. Al fer servir màquines amb poca capacitat, s'ha d'intentar optimitzar aquest procés. Sinó, els temps de resposta no poden ser els desitjats. Això implicaria que la màquina es podria quedar sense la memòria suficient per poder utilitzar o desar posteriorment informació.
- Fàcil integració. Donat que el projecte al qual s'ha d'aplicar aquest SCA ja està molt avançat, el fet de provocar molts canvis pot tenir una gran repercussió. El disseny de l'actual *aDTN* és complex, per tant, qualsevol canvi pot alterar o provocar modificacions a molts mòduls o components. Una integració complexa pot comportar molt temps de dedicació. Al mateix temps també podria comportar molts càlculs, afectant també a la condició anterior.
- Capacitat de modulació. Per a que el procés d'integració sigui senzill, aquest sistema ha de poder ser separable en diferents funcions. No sempre l'execució del codi es fa al mateix moment en que es rep la informació. Una capacitat de modularitat dins del projecte, ajuda a que es pugui executar el codi en diferents moments o punts. D'aquesta manera també s'aconsegueix que el treball es pugui adaptar a altres projectes.

Donades aquestes condicions, el un dels SCA que les compleix trobat ha sigut *ABAC*.

Al escollir *ABAC* com a sistema per poder fer el control d'accés, però ser un sistema que no s'adequa del tot als requeriments establerts, aquest va patir una sèrie de canvis durant l'execució d'aquest objectiu.

El xifratge del codi per la seva posterior utilització és una altra de les decisions que es varen prendre durant el transcurs del treball.

Dintre de les possibilitats trobades pel xifratge, es varen descartar algunes d'elles, com per exemple, el parell de claus pública/privada. Aquest sistema requereix una quantitat de processament més elevat que altres sistemes. Al mateix temps, també necessita que les claus públiques estiguin emmagatzemades a algun servidor, i aquest servidor ha d'estar disponible en qualsevol moment, però en una aDTN potser no és possible.

També es podria haver desat totes les claus a tots els nodes, així no faria falta les connexions amb el servidor. Però aquesta solució té dos inconvenients.

El primer és que les màquines amb les que es treballa són *Raspberry Pi* [6], el que significa que la capacitat d'emmagatzemament és escassa, i s'ha d'intentar d'aprofitar al màxim.

El segon inconvenient és que al tractar-se d'una *aDTN* en la que qualsevol pot participar, hi ha moltes probabilitats de que els nodes no tinguin les claus dels últims nodes que hagin entrat a la xarxa, per tant aquests no podrien participar. S'hauria de compartir totes les claus cada un temps. Per tant, implicaria tenir una connexió constant amb la resta de la xarxa, cosa que pot se que no existeixi.

El *hash* escollit per realitzar el resum del codi a executar ha sigut *SHA256* per les següents raons:

1. Té una probabilitat de col·lisió tant baixa que es considera que és 0. Així s'aconsegueix que el càlcul del *hash* de dos codis no pugui donar el mateix resultat.
2. Els càlculs necessaris són pocs, per tant, les màquines de la xarxa el realitzaran ràpidament. S'ha de tenir en compte aquesta opció donat que els dispositius no tenen una gran capacitat de càlcul.

Així, al només fer un resum del contingut del codi, s'aconsegueix que ocupi menys espai dins del disc. El resum sempre ocupa el mateix espai en disc (64 Bytes).

3.3 Disseny

Dels dos mecanismes que té *ABAC* per realitzar les consultes, *PUSH* és l'utilitzat al disseny i la resta del TFG. El motiu és que s'està aplicant aquest sistema a una xarxa que ha de permetre retards i pèrdues de connexió. En el model *PUSH* les polítiques estan situades dins de les màquines. Si es perd la connexió amb la resta, es pot seguir amb l'execució, ja que conté tota la informació que necessita. L'independència d'aquest és el factor clau per a prendre la desició.

L'altre opció, *PULL*, necessita l'interacció amb el servidor de manera constant, i podria no ser accessible en el moment de realitzar la consulta. Aquest model queda exclòs al no ser viable dins d'una xarxa *DTN*, per culpa de la seva dependència amb la xarxa.

Seguidament s'expliquen les principals modificacions que es van realitzar al model *ABAC*, per aconseguir una adaptació a l'entorn on es va aplicar. Si no es s'haguessin realitzat aquestes modificacions, la base existent del model no hagués sigut acceptable donats els recursos disponibles.

La primera modificació important que es va fer va ser la comunicació entre el *PEP* i el *PDP*. El model *ABAC* comunica els dos elements, de tal manera que el *PEP* envia la sol·licitud al *PDP* per si pot accedir o no al recurs, i aquest respon a l'acte. Aquesta comunicació s'ha eliminat.

L'explicació de la modificació resideix en el fet de que aquests dos processos, seguint el model de l'*aDTN,* no estan directament comunicats, sinó que treballen per separat. El *PEP* és cridat per la funció que controla l'emmagatzemament del codi al fitxer, aquesta funció està situada dins de *BundleManager*, i al fitxer l'anomenarem *codi.c.* No fa cap crida contra el *PDP* en cap moment, ja que l'execució no es fa en el mateix instant.

El *PDP* forma part del *RIT*. Cada cop que s'intenta executar una funció que té relació amb el *RIT*, ja sigui un *GET*, *SET* o altres, sempre consulta al *PDP* si pot realitzar l'acció. Així ens assegurem de que l'acció és controlada i compleix les regles definides dins l'*ACL* (*Access Control List*).

La segona modificació important del model involucra al servidor, que ja no consulta al *PEP* si pot realitzar la sol·licitud. El *PEP,* seguint amb la primera modificació, ja no està involucrat amb l'execució del codi.

Després de la segona modificació, el *BundleManager* simplement guarda el codi amb un afegit del *PEP* en al fitxer *codi.c*, i el *RIT manager* s'encarrega posteriorment de la seva execució.

Per a poder identificar l'accés de cada codi un cop el *bundle* arriba, el node farà una crida al *PEP*. Aquest afegirà una línia de codi amb un número aleatori i el hash del codi sencer a un fitxer, que anomenarem *hashTable.txt*, que el *PDP* farà servir posteriorment.

Aquest mateix número també s'afegirà dins del codi (després de fer el *hash*, d'aquesta manera no serà cada cop diferent) com a un *#define* de C. Aquest ha de ser de la variable R, que al fer una crida a qualsevol acció referent al *RIT* anirà com a últim paràmetre, quedant, per exemple, de la següent manera: *set(codi,R);*.

D'aquesta manera, a partir de la segona modificació d'*ABAC*, els programadors del codi que circularà per la xarxa han de tenir present aquest canvi. A partir de l'integració, tot el que estigui relacionat amb el *RIT*, haurà de tenir un paràmetre més al final que l'establert abans d'afegir aquest TFG a l'*aDTN*.

El fitxer anteriorment anomenat *hashTable.txt*, conté el número aleatori associat amb el *hash* del codi, separats per un espai. Aquest fitxer només pot ser accedit pel *PEP* i el *PDP*. Serveix per afegir un grau d'aleatorietat al codi, d'aquesta manera hi ha menys possibilitats de ser víctimes d'un atac. També s'ha de fer notar que les dades transferides entre els diferents mòduls seran inferiors, ja que el número aleatori ocupa menys que el *hash* del codi.

El fitxer que farà d'*ACL* l'anomenarem *ACL.txt*. Contindrà el *hash* del codi, un espai en blanc i, seguidament, separat per comes, la llista d'accions que aquest *hash* pot realitzar. Aquest fitxer només pot ser consultat pel *PDP*.

El disseny inicial del projecte és molt simple. Seguint la metodologia evolutiva, simplement es composa de la connexió entre els diferents mòduls, que es faran servir al llarg del projecte. Fins el moment, la resposta del *PDP* serà sempre positiva, i es farà una comprovació del seu correcte funcionament.

La segona part del disseny, ampliant una mica l'anterior, es composa del càlcul del *hash* al *PEP* i *PDP*, i la posterior comprovació de que ha estat correcte aquest càlcul.

Al mateix temps, el mòdul *PDP* consultarà el fitxer *ACL.txt*. Conté el *hash* i les accions que pot executar aquest codi, i mostrarà el llistat de les accions que pot realitzar per pantalla. En aquest procés, es podrà veure si realment s'està consultant el fitxer correctament.

En resum, el node ha d'enviar el codi, s'han de realitzar les accions explicades anteriorment (creació dels fitxers, consultes i altres), tenint en compte que el *PDP* ja està dins del *RIT.* De tal manera que ja pot executar o no (segons l'*ACL)* la comanda que s'ha sol·licitat.

Seguidament es pot veure un diagrama de seqüència amb les accions que es faran durant el procés. Només il·lustra el procés que es fa fins que es desa el codi al *fitxer.c*.

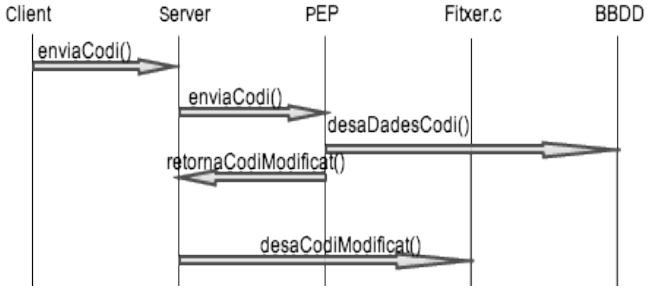

Fig2. Diagrama de seqüència del procés per desar el codi que posteriorment s'executarà.

La funció `enviaCodi()` és en realitat el procés que fa el client per enviar el *bundle* cap al *Server*, durant el temps que estan connectats, per posteriorment poder ser executat. Aquest últim, abans de desar-lo al fitxer o BBDD, fa un procés nou, que és enviar aquest cap al *PEP*. Un cop el codi ja està enviat, el client ja pot desconnectar del servidor, ja que al ser una tasca asíncrona, el codi no té per a que executar-se al mateix moment en el que s'envia.

La funció `enviaCodi()` que va des de *Server* cap a *PEP*, pot ser realitzada encara que ja no es tingui connexió amb altres nodes. Aquest fa una crida al *PEP* passant-li el codi que ha rebut del client, per que el *PEP* el modifiqui. Afegeix una línia que defineix el número aleatori, també creat pel *PEP*. Al mateix temps, crea el *hash* del codi abans d'afegir-li la línia, i afegeix aquest *hash* juntament amb el número aleatori a un fitxer que fa de BBDD, de tal manera que quedin associats el *hash* amb el número aleatori, per ser consultat posteriorment pel *PDP*.

El *PEP* retorna cap al Server el codi modificat amb la línia que afegeix el valor del nombre aleatori. Per a que aquest pugui desar-ho, fent servir el *BundleManager*, per desar el codi a un fitxer per ser executat més tard.

Un cop el servidor consideri que ja es pot executar el codi, és a dir, que no hi hagi cap transferència en el moment, l'executa i es troba durant aquest procés que hi ha accions que requereixen alguna interacció amb el *RIT.*

Ja sigui simplement consultar, modificar, eliminar o inserir informació en aquest.

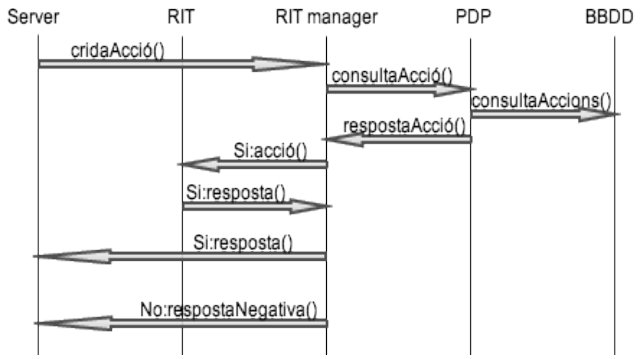

Fig3. Diagrama de seqüència del procés per a executar el codi del client.

Per a fer-ho, en el procés de `cridaAcció()` que va des del servidor cap al *RIT manager*, el qual és l'intermediari entre servidor i *RIT*, executa l'acció, la qual conté, apart dels valors descrits a la *API*, el valor numèric aleatori.

A l'intentar fer l'acció, crea una consulta al *PDP*, que conté un codi estipulat indicant quina acció vol consultar i el número aleatori afegit anteriorment.

Agafant el número aleatori, fa una consulta al fitxer de BBDD creat pel *PEP*, per saber quin és el *hash* associat al número. Aquest *hash* es fa servir en operacions posteriors. El número aleatori està inclós per evitar que qualsevol pugui executar les accions, al crear el *PEP* el número i insertant-lo un cop ja no és accessible pel client.

Un cop el *PDP* té el *hash* del codi que es vol executar, fa una consulta a una altra BBDD, anomenada *ACL* que simplement conté el *hash* i seguidament, separat per un espai en blanc, el llistat d'accions que pot executar aquest codi. Implica que si una acció no està inclosa en aquesta llista, no la pot executar. Les accions que estan dins de la llista van separades per una coma.

Quan el *PDP* disposa de la llista d'accions, simplement fa una comparació per saber si la sol·licitada està dins de la llista. Quan ja ha fet aquesta comparació, pot retornar dues respostes, positiva o negativa. El *RIT manager* rebrà aquesta resposta, i podrà realitzar dues accions:

`Si:acció()` farà directament l'acció contra el *RIT*, ja sigui consulta, inserció, modificació o eliminació, i quan obtingui la resposta d'aquest, la retornarà cap al servidor, sent així com funcionava abans d'aplicar el SCA.

`No:respostaNegativa()` és la nova opció que s'afegeix. Aquesta significa que el codi no està autoritzat a realitzar l'acció sol·licitada. Quan es dona aquesta resposta, és responsabilitat del programador del codi atendre a les conseqüències, ja sigui continuant l'execució evitant aquesta informació o bé aturant l'execució del programa, per exemple.

3.4 Implementació

Seguint la metodologia evolutiva i els passos que s'expliquen a l'apartat 3.3, no hi ha gaire lloc a implementacions variades. Una d'aquestes variacions és la que es mostra a continuació, que presenta la programació del codi que es va fer servir per realitzar aquesta tasca.

Primer de tot, el projecte del grup SeNDA ha estat realitzat amb el llenguatge C, afegint algunes biblioteques pròpies per facilitar la feina. Aquestes biblioteques principalment fan referència al tractament dels *bundles*, i les accions possibles a fer amb el *RIT*.

Una d'aquestes llibreries que fa referència al *RIT*, va ser modificada per poder realitzar les tasques d'una manera més ràpida.

Seguint el disseny, primer de tot es van de crear els dos mòduls, *PEP* i *PDP*. La comunicació entre els dos es va fer mitjançant *sockets*.

Es van escollir *sockets* ja que, per fer proves per futures aplicacions en les que els dos mòduls estan separats en diferentes màquines, era un sistema millor que no pas fer crides. Al mateix temps, si funcionava enviant missatges, considerant que eren màquines diferents, el funcionament sent la mateixa seria més ràpid i còmode. Per a aconseguir-ho, es va crear un mòdul. La funció d'aquest mòdul era enviar al *PEP* un missatge, el qual contenia un codi. El *PEP* feia una crida al *PDP*, que en el moment encara estava buida, i aquest enviava una resposta, sempre positiva, fins el retorn al mòdul inicial.

Fig4. Esquema del mòduls i les seves comunicacions per la primera part de la implementació.

Per la segona part del disseny, s'havia de buscar, sabent que ja havia estat creat anteriorment i és codi lliure, un codi per fer el *hash* amb *SHA256* del algoritme que ens havia enviat el *Server*. Aquest mòdul per fer el *hash* es va acoblar als diferents mòduls que l'utilitzarien. Un cop acoblat el nou mòdul, tant al *PEP* com al *PDP*, ja es van poder realitzar les comprovacions del *hash* del codi del resultat, amb el donat per alguna pàgina de confiança.

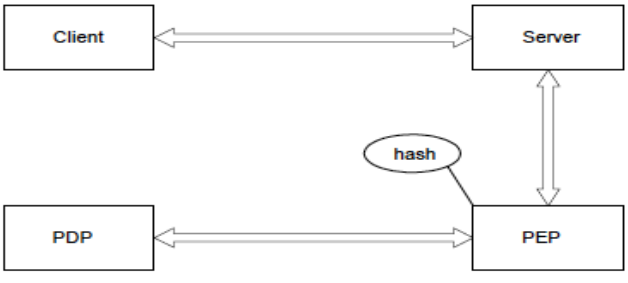

Fig5. Esquema del mòduls i les seves comunicacions per la segona part de la implementació.

La tercera part, un cop afegides les biblioteques per crear el número aleatori, i per aconseguir una millor aleatorietat, amb una variable de temps incorporada, consistia en inserir el nombre aleatori, juntament amb el *hash*, dins d'un fitxer anomenat *hashTable.txt*.

Per una consistència millor durant de l'execució, es van fer totes les comprovacions sobre l'existència del fitxer, abans de la interacció amb aquest.

Posteriorment es s'enviava un missatge al *PDP*, amb l'ordre de buscar el codi. El mòdul es quedava amb el nombre aleatori per aconseguir el *hash* del codi a consultar. Posteriorment consultava l'*ACL* per saber les accions que podia complir aquest, ensenyant-les per pantalla.

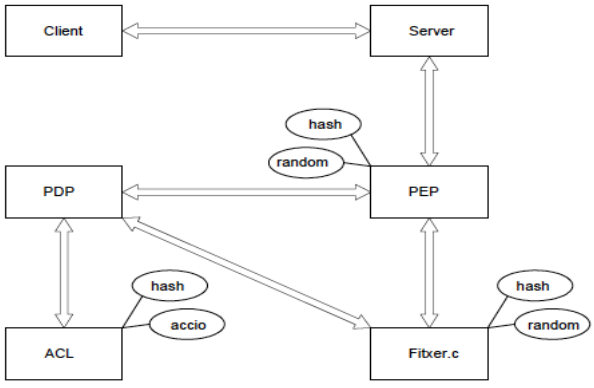

Fig6. Esquema del mòduls i les seves comunicacions per la tercera part de la implementació.

La quarta i última part del disseny va ser l'acoblament del *PDP* dins del *RIT manager*.

Durant l'execució del codi que havia enviat el *Client*, el qual el *Server* l'havia desat en el fitxer *codi.c*, cada cop que es feia una crida cap a una de les accions del *RIT manager*, havien de ser controlades pel *PDP*, mitjançant les accions permeses, establertes a l'*ACL*. Aquestes regles tenien que estar anteriorment introduïdes dins el fitxer, seguint l'estructura descrita al disseny.

Les accions dins del *RIT* tenen una nova condició respecte el codi original, afegida en aquesta última part. Aquesta condició conté el tipus d'acció que es vol fer contra el *RIT*, i el número aleatori donat anteriorment.

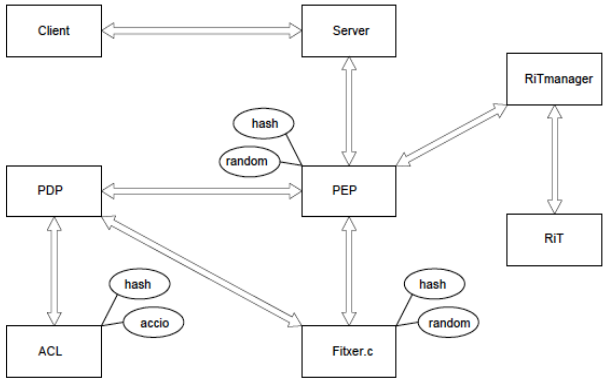

Fig7. Esquema del mòduls i les seves comunicacions per la quarta part de la implementació.

La condició fa la crida cap al *PDP* per obtenir la resposta sobre l'acció.

Si la resposta és positiva, es pot seguir amb el codi, en canvi, si és negativa, segueix sense aquesta acció.

3.5 Integració

Aquesta objectius del TFG simplement es basa en la integració d'aquest al desenvolupat pel grup SeNDA. El disseny ja ha estat fet per a que aquest pas sigui el més senzill possible. Afegint unes poques línies de codi i els fitxers que contenen els mòduls, és suficient pel correcte funcionament de la SCA dins l'esquema.

Per part del *PEP*, els mòduls que controlen la inserció del codi al fitxer que posteriorment hauran de ser executats, hauran d'incloure, abans de crear el fitxer amb el codi la crida al mòdul *PEP*, per la modificació del codi i afegint el número aleatori, i la creació del fitxer amb el *hash* i el número.

Per part del *PDP*, dins del fitxer que controla les crides *RIT manager*, a cada funció que faci una crida cap a ell, s'ha d'afegit una condició, en aquest cas es va escollir un *if*, que fa una crida a la funció del mòdul *PDP*, amb el nombre aleatori proporcionat. En cas de que es compleixi la condició, segueix amb el codi, en cas contrari, s'obvia el resultat de la funció.

4 Proves i Resultats

La present secció està dedicada a la mostra de l'efectivitat del problema resolt.

Primer de tot es mostra les proves realitzades per comprovar el funcionament correcte. D'aquesta manera s'assegura que les respostes siguin sempre les esperades.

A continuació es mostren els resultats dels càlculs dels temps d'execució. Així es demostra que l'integració del TFG dins d'*ABAC* no comporta una gran pèrdua de temps ni recursos.

4.1 Proves

Els resultats de les proves han sigut els esperats, tant per la part teòrica com per la part pràctica. Per la part teòrica, tot i que el disseny ha sofert alguns canvis des de l'inici, el plantejament general ha estat ben encaminat. Per la part pràctica, a excepció d'algunes comandes o funcionalitats específiques, no han aparegut complicacions, ja que la majoria de funcions utilitzades ja eren conegudes.

S'ha anat millorant i ampliant l'aplicació per realitzar els objectius, al principi programant d'una manera molt específica per l'aplicació que s'està realitzant, i més endavant, adaptable per futures aplicacions en altres sistemes, ja que cada funcionalitat està separada de la resta.

Al realitzar les proves, com s'esperava des del principi, la capacitat de processament necessària per realitzar els càlculs requerida és assumible per aconseguir temps de resposta que permetin la transferència de dades, complint els requisits d'*aDTN*, per tant, en les màquines de les que disposa el projecte es podran executar correctament i sense aportar problemes de falles durant la transmissió de la informació.

Descrivim ara les proves que es van realitzar durant el desenvolupament de l'aplicació, seguint les accions realitzades, ja que van completament lligades a la programació.

Al principi de tot, les proves van ser mínimes, al ser un disseny bàsic en que l'única funció serà la de fer una crida a la funció, o calcular el *hash*. Cap al final del projecte, la quantitat de proves i la complicitat d'aquestes augmentarà, pel que s'ha d'incrementar l'esforç dedicat a aquestes.

Per la primera part del disseny, les proves es van realitzar fent comunicacions entre els mòduls, fent crides a les funcions de cadascun, enviant alguna informació i mostrant els resultats per pantalla.

Per la segona part del desenvolupament del projecte, la prova a realitzar va ser la comprovació de que el *hash* resultant de l'operació fos l'esperat, comparant amb el resultat d'algun càlcul de confiança.

Les proves realitzades per la tercera part, van ser mirar que els nombres aleatoris no fossin els mateixos, seguits o seguissin cap patró calculable. Per això anteriorment es va afegir la variable del temps. També es va supervisar el procés d'introducció tant del *hash* com del número aleatori al fitxer per part del *PEP*. Per part del *PDP*, es va comprovar que els resultats mostrats per pantalla corresponguessin als del fitxer de l'*ACL*. Al mateix temps que es va mirar que la resposta del *PDP* fos l'esperada (en aquest pas encara era sempre positiva) i que el mòdul inicial la rebés correctament.

Per l'última part, al intentar executar el codi que s'havia emmagatzemat anteriorment a un fitxer, el *RIT* havia de fer una comprovació al *PDP* per mirar si podia realitzar l'acció o no. Les proves a realitzar van ser totalment lligades a la resposta que havia de donar aquest últim mòdul, és a dir, positiva o negativa. Sabent les respostes, incloses o no al fitxer de l'*ACL*, fer que el codi contingués una acció amb resposta positiva, i una amb resposta negativa.

4.2 Resultats

Per comprovar la magnitud de temps en que els nous mòduls afegits alteraven les respostes, es va calcular el temps que es tardava en fer una sèrie de consultes dirigides al *RIT.*

Primer de tot, abans de mostrar els resultats, s'ha de comentar que aquests van ser calculats amb una comunicació entre mòduls fent servir *sockets.* Això implica que els temps de resposta són més lents que si es fessin directament sobre una sola màquina.

Per realitzar aquesta prova, des d'un node client s'envien 10.000, 20.000 i 50.000 peticions contra el *RIT*. Les consultes són un *GET*. Es considera que sempre seran positives, ja que si no fos així, no tindria cap sentit. La raó de que no tingui cap sentit és que sense SCA, és a dir, abans d'integrar el TFG, no podien apareixer respostes negatives. Això influiria en els resultats finals.

S'ha intentat buscar en codicions iguals en les dues proves, afegint que, quan la búsqueda es fa amb SCA, aquesta no està a la llista de l'ACL, provocant així que es tingui que buscar a tot el fitxer.

Aquestes proves han sigut realitzades a màquines virtuals, emulant també a les característiques dels dispositius als que seran instal·lats. Les característiques es poden trobar a la pàgina web de *Raspberry Pi*.

S'ha fet aquesta simulació per acostar-se al màxim a l'entorn en el que futurament estarà situat TFG, i al mateix temps per aconseguir realisme als resultats obtinguts.

Els resultats han estat calculats extreient els temps màxims i els mínims, i fent la mitja del restant. D'aquesta manera s'aconsegueix una aproximació més general.

La següent taula mostra els resultats obtinguts durant les proves:

Repeticions	Sense SCA	Amb SCA
10000	23,01s	23,53s
20000	46,19s	47,18s
50000	115,6s	118,56s

Taula1. Mostra dels resultats obtinguts a l'executar la comanda repetides vegades dins l'entorn, mesurades en segons.

Com es pot veure a la Taula2, els resultats no varien gaire entre els que utilitzen SCA i els que no. Els resultats s'han obtingut amb la comanda `time` de Linux.

Aquesta comanda mostra el temps total que s'ha tardat en executar la comanda que va seguida de `time`.

Una de les raons de la similitud dels resultats és que, com mostra la pròpia comanda `time`, quan es fa servir el SCA, també s'utilitzen recursos del sistema per poder optimitzar el temps. D'aquesta manera, no només utilitzant els recursos assignats a l'execució del codi, s'aconsegueix que la variació del temps entre les dues opcions sigui assumible.

Un dels problemes a partir la tercera part del disseny és que l'escalabilitat de les accions a realitzar no existeix, al realitzar la consulta al *PDP* afegint una condició a cada possible acció, l'automatització d'aquesta no existeix, tenint que posar un codi especial a cada nova acció, i alertant als creadors de codis que han de canviar aquest fet.

5 Conclusions i línies futures

Per acabar, a continuació es mostren les conclusions i les possibles linies futures que pot assolir el TFG.

Dins de les conclusions s'exposen les millores tant a nivell personal com, després de la seva incorporació, les millores realitzades a l'*aDTN*.

A la secció de linies futures, es mostren algunes de les variacions o ampliacions que es podrien realitzar per aconseguir millors resultats.

5.1 Conclusions

Els objectius proposats a l'inici de l'article es van aconseguir resoldre amb un gran grau d'assoliment. Des de la cerca d'informació, fins la integració del TFG.

El projecte ha estat profitós personalment en quant a definició del problema, cerca de solucions i presa de decisions. Per la primera part, ja definida pels professors que varen proposar el TFG, s'ha hagut d'observar l'entorn en el que estava situat el problema. Un cop detectat i

situat, es va iniciar la cerca de possibles solucions. Aquesta cerca simplement va servir per trobar vàries maneres de resoldre el problema, que satisféssin el plantejament inicial.

Amb el llistat de possibles SCA que fossin una solució viable, va arribar el moment més delicat del procés. La presa de decisions, quin model s'escull, per a què, com es modificarà pel correcte funcionament, i sobretot, intentar de que sigui el definitiu i no s'hagin de realitzar canvis durant els processos posteriors.

Amb les decisions preses fent servir les característiques i necessitats del problema, la implementació d'una *PEP* que modifiqués el codi dels *bundles*, afegint una variable amb un valor aleatori i el *hash* del codi (abans de afegir el valor de la variable) a un fitxer, i més endavant, durant l'execució d'aquest codi, el gestor del *RIT* fés una crida al *PDP* per, consultant el valor al fitxer que ha introduït el *PEP*, i amb una *ACL*, verificar si el codi està autoritzat o no.

Al ser més crides des del codi ja existent dins del projecte, podria suposar una pèrdua de recursos i temps. Tot i així, l'esforç per la seva integració dins del projecte va ser mínim, gràcies a un disseny que ja tenia en compte aquest inconvenient. D'aquesta manera també s'aconsegueix que, amb la modularitat aconseguida, es pugui adaptar a altres projectes, afegint la mateixa quantitat de codi a cada projecte.

Com s'ha mostrat a l'apartat de Resultats, el fet d'afegir aquests mòduls i els nous càlculs al sistema, el temps de resposta no es veu gairebé alterat. Això ajuda a que el canvi, respecte dels usuaris, sigui imperceptible.

Al haver-hi una petita diferència entre els temps, el projecte es viable en quan a la tecnologia aplicada.

5.2 Linies futures

Per acabar de completar el treball, en línies futures es podria afegir seguretat al moment d'accedir als fitxers, d'aquesta manera no es podrien modificar a excepció de l'administrador, per exemple.

Al mateix temps, es podria afegir un altre sistema de control d'accés per complementar *ABAC*. Aquest sistema podria estar basat en *RBAC*. Així s'aconseguiria que també es poguessin limitar els accessos a informació depenent del rol del sol·licitant.

Com a possibilitat de linia futura també es pot afegir la modularitat del *PDP* respecte el *RIT*. D'aquesta manera, es podia aconseguir que altres aplicacions poguessin fer servir aquest mòdul, per accedir a altres fonts d'informació. Així també s'aconseguiex un reaprofitament del TFG per altres aplicacions.

Agraïments

El grup SeNDA ha ofert una gran ajuda a realitzar aquest projecte, tant en material com en suport conceptual o moral.

També a tota la gent que ha donat suport moral per portar endavant el TFG, incloent familia i amics.

Bibliografia

[1] V Cerf, S Burleigh, A Hooke, L Torgerson, R Durst, K Scott, K Fall, and H Weiss. "Delay-tolerant networking architecture". RFC 4838 (Informational), 2007.

[2] C. Borrego, S. Robles, A. Fabregues, A. Sánchez-Carmona, "Application-specific routing diversity in Delay and Disruption Tolerant Networking using mobile-code", Univ. Autònoma de Barcelona, Bellaterra. Submitted 2014.

[3] Ravi Sandhu, "The Future of Access Control: Attributes, Automation an Adaptation", The Institute for Cyber Security (19 Juny 2013).

[4] empowerID, Identity Management (2013). Authorization Services. Role-Based and Attribute-Based Access Control [Online]. Diponible: http://www.empowerid.com/products/authorizationservices. Última consulta: 2/2/2014.

[5] Axiomatics, Policy Enforcement Points [Online]. Disponible: https://www.axiomatics.com/policy-enforcement-points.html. Última consulta: 2/2/2014.

[6] Axiomatics, Policy Decision Points [Online]. Disponible: https://www.axiomatics.com/policy-decision-points.html Última consulta: 2/2/2014.

[7] Raspberry Pi [Olilne]. Disponible: http://www.raspberrypi.org/. Última consulta: 2/2/2014.

[8] Security of Networks and Distributed Aplications (SeNDA), Universitat Autònoma de Barcelona. Bellaterra. [Online]. Disponible: senda.uab.cat. Última consulta: 2/2/2014.

[9] "Delay Tolerant Networking Research Group. Delay tolerant networking research group website", (Desembre 2013). [Online]. Disponible: http://www.dtnrg.org. Última consulta: 11/2/2014.

[10] Michael Cooney. "Nasa exploring groundbreaking space network to sustain large data dumps and trips to the moon, mars", (3 Novembre 2013). [Online]. Disponible: http://www.networkworld.com/community/blog/nasa-exploring-groundbreaking-space-network-sustain-large-data-dumps-and-trips-moon-mars. Última consulta: 11/2/2014.